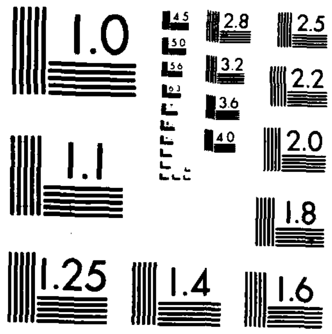

1.0
4.5
5.0
5.6
6.3
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

(4)

ISI Research Report
ISI/RR-88-195
April 1988

University of Southern California

Yu-Wen Tung
Christian Matthiessen
Norm Sondheimer

On Parallelism and the Penman Natural Language Generation System

INFORMATION SCIENCES INSTITUTE ISI 213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

ISI Research Report
ISI/RR-88-195
April 1988

University of Southern California

Yu-Wen Tung
Christian Matthiessen
Norm Sondheimer

On Parallelism and the Penman Natural Language Generation System

INFORMATION SCIENCES INSTITUTE
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported by the Defense Advanced Research Projects Agency under Contract No MDA903-81-C-0335. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT This document is approved for public release, distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) ISI/RR-88-195	5. MONITORING ORGANIZATION REPORT NUMBER(S) ---------------
6a. NAME OF PERFORMING ORGANIZATION USC/Information Sciences Institute	6b OFFICE SYMBOL *(If applicable)*
7a. NAME OF MONITORING ORGANIZATION ---------------	
6c ADDRESS *(City, State, and ZIP Code)* 4676 Admiralty Way Marina del Rey, CA 90292	7b. ADDRESS *(City, State, and ZIP Code)* ---------------
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA	8b. OFFICE SYMBOL *(If applicable)*
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA903-81-C-0335	
8c. ADDRESS *(City, State, and ZIP Code)* DARPA 1400 Wilson Blvd. Arlington, VA 22209	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
---------------	---------------	---------------	---------------

11 TITLE *(Include Security Classification)*
On Parallelism and the Penman Natural Language Generation System [Unclassified]

12 PERSONAL AUTHOR(S)
Tung, Yu-Wen; Matthiessen, Christian; Sondheimer, Norm

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Research Report	FROM ______ TO ______	1988, April	

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
chooser & inquiry framework, classification, generation algorithm, grammar, inquiries, KL-TWO, knowledge representation, natural language generation,
(continued on other side)

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This report discusses parallel processing for the Penman natural language generation system. We first analyze the computational requirement of the generation process. We then identify aspects of this computation that could benefit from being carried out in parallel.

The Penman generator is composed of a systemic grammar, the Nigel grammar, and its environment. These two components are functionally separated and interface to each other via an inquiry mechanism. This implies that Nigel and its environment can be processed in a distributed way. We also illustrate how both Nigel and the major part of its environment, the KL-TWO knowledge base, can each be processed in parallel.

(continued on other side)

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Sheila Coyazo Victor Brown	22b. TELEPHONE *(Include Area Code)* 213-822-1511
22c. OFFICE SYMBOL	

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

18. (continued)
Nigel, NIKL, parallel processing, Penman, systemic functional grammar, text generation

19. (continued)

In the Nigel grammar, the systems, choosers and realization statements can be activated simultaneously according to some computational dependency that resembles the system network. The KL-TWO knowledge base can be implemented as a parallel computing system and two existing approaches, using Classifier Systems and Connectionist Models respectively, are analyzed and assessed.

Table of Contents

List of Figures

1. Introduction

This paper explores the feasibility and benefits of parallelism in the Penman text generation system developed at USC/ISI [Mann 83a]. More specifically, it examines the opportunities for parallel processing in the grammatical component and in the knowledge representation component of Penman.

Penman uses Halliday's systemic grammar [Halliday 76] as its framework to generate English sentences that conform to an *a priori* intention and plan to communicate [Mann 83b]. The computation of Penman is complex by its nature -- it requires non-trivial graph search and matching of patterns from a large knowledge base, and active coordination of many kinds of information having independent origins and character. The central question is if such computation can be made more effective. Let's first look at the overall organization of the Penman system.

1.1 The Penman System

We will consider two major components of the Penman generator: Nigel, the grammatical generator, and KL-TWO, the knowledge representation system.[1]

Nigel is a grammatical component that accepts an input language based on first-order logic,[2] and generates English sentences that reflect the meaning of the input one at a time. It consists of a systemic-functional grammar and an interface between the grammar and the rest of the Penman generator, which can be thought of as Nigel's environment in generation. One of the major components in this environment is the KL-TWO knowledge base. The outline of the organization is given in Figure 1-1. It represents only the knowledge resources, not the organization of the generation processes; the arrows indicate which components interact directly with one another. The details will be explained presently.

It is important to note that the organization of Nigel factors the task of sentence generation into two different kinds of processes: the interface and the grammar.

The Nigel interface interacts with the environment by presenting it with inquiries for information ([Mann 83b], [Matthiessen 88]); the information is supplied as responses to these inquiries from the environment. Most of the responses to these inquiries come

[1]We will not discuss other components of Penman, such as the text planner. For an early overview of the Penman design, see [Mann 83a].

[2]This language is called *Meaning Representation Language*, or *MRL*. Each *MRL* statement corresponds to the propositional and speechact contents of a natural language sentence.

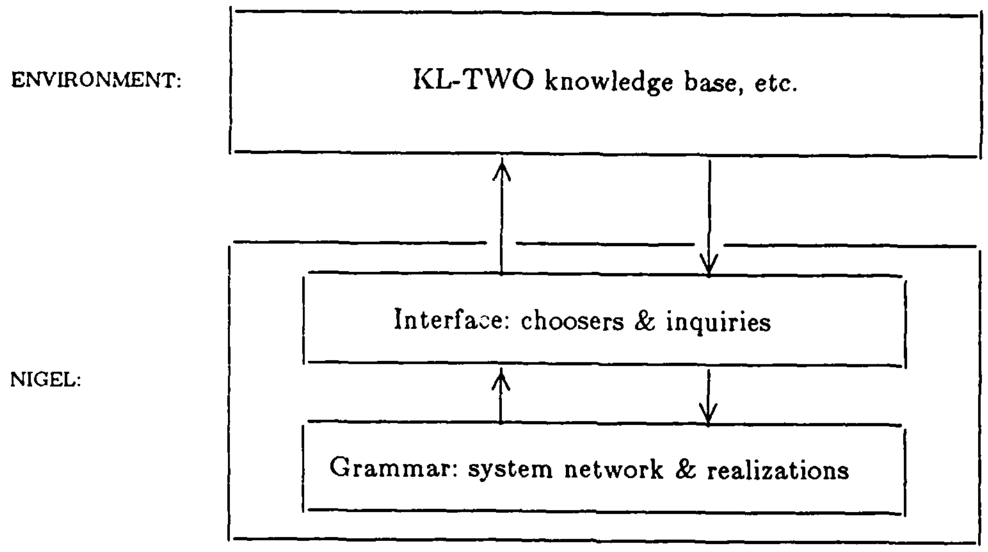

ENVIRONMENT:
KL-TWO knowledge base, etc.
NIGEL:
Interface: choosers & inquiries
Grammar: system network & realizations

Figure 1-1: Nigel and its environment in Penman

from the KL-TWO knowledge base.[3] KL-TWO has two components: NIKL and PENNI. NIKL is a definitional (terminological) component. It has definitional knowledge of "concepts" and "roles," and it provides limited quantificational reasoning. PENNI has an assertional component RUP, which contains a database of propositional assertions and provides propositional reasoning capability. PENNI also has a capability of interfacing RUP with NIKL. It exploits both the assertional component and the definitional component to respond to Nigel's inquiries. In other words, Nigel's environment (including KL-TWO) serves as an oracle for Nigel to consult.

1.2 Overview of Parallelism in Penman

We can now return to the central question of how the Penman computation can be made more effective. Penman's text generation process is complex, but it can be broken down into the two independent components we identified above: the Nigel component and its environment. Since these two components communicate with each other by exchanging messages only -- the messages are inquiries and answers to the inquiries --

[3]Nigel inquiries are also rely on information from, for example, the discourse model of the Penman design.

and Nigel does not need to know how the answers are computed, the Penman system can be viewed as a distributed system consisting of two components and a message-passing interface.

This distributed system does not in itself save any computation time, since the computation only alternates between the two components. To gain some speed-up, faster components or multiple components, or both, will be required. For example, we could use several slower components to balance the computation speed of the faster one so that the overall system is faster. We could do this by treating each component as a black box with unknown internal organization. This is the coarsest-grained parallelism.[4]

We could apply the same black-box philosophy to each component, and perhaps also their subcomponents. However, whether or not the Penman system can be benefited from finer-grained parallelism is not a trivial question. The answer depends on the computation constraints due to the dependency relation between one component and another, and it also depends on the communication overhead due to the interaction among components in a parallel computation environment. These all require, of course, a good understanding of the computational requirement of the Penman system.

1.3 Organization of the Paper

In the following sections, we show how Nigel and the knowledge base KL-TWO can each be executed in parallel.

Section 2 examines how Nigel "computes" to generate an English sentence. Section 3 proposes four kinds of parallelism applicable to Nigel. We also analyze the benefits and limitations of each kind of parallelism, and offer a real example in this section. Section 4 examines how KL-TWO works. And Section 5, we study how KL-TWO computation can be done in parallel and provides a review of existing work in this area. This includes a Classifier Systems approach taken by Forrest, and a Connectionist Models approach taken by Derthick.

Finally, we indicate a future direction of this research.

[4]The efficiency of solving a problem by breaking it down to pieces relates directly to the parallelism inherent in the problem, and the parallelism can be characterized by the size of the pieces -- or so called granularity. A problem may have coarse-grained parallelism, meaning that the problem can be broken down into some concurrent jobs, tasks or processes. And a problem may have fine-grained parallelism, meaning that the problem can be broken down into some concurrent instructions or data operations.

2. The Nigel Component

This section describes the computation of each of the Nigel components. We begin with the system network of the grammar component, since it provides the global organization of the Nigel component and of the grammatical generation process. We will then present the mechanism for expressing selections from the system network as specifications of grammatical structure, so-called realization statements, and then the chooser and inquiry interface that relates the system network to the environment of the Nigel component. After this brief overview of the resources of the Nigel component, we will discuss the process of traversing the system network in generation. Finally, we will discuss the organization of a chooser in some more detail.

2.1 Grammar: The System Network

The system network in Nigel is the fundamental organizing principle of the grammar, as it is for systemic grammars in general. It is a large directed acyclic graph whose nodes are choice points, called systems. The system network also serves to organize the generation process and the semantic interface, and it defines the contexts in which structure building realization statements are applied. An excerpt from the system network of TRANSITIVITY and MOOD can serve as a simple example, as shown in Figure 2-1.

This system network consists of eight systems, whose names are capitalized. Each system has two or more terms or **output features** representing a grammatical alternation, such as "indicative/imperative" in the system MOOD TYPE. In addition, each has an **entry condition**, such as the feature "clauses" serving as the entry condition of the system MOOD TYPE. The entry condition may be a simple feature (such as clauses in the systems MOOD TYPE and interrogative in INTERROGATIVE TYPE) or a complex of features. A feature complex may be a disjunction of features (such as the disjunction of declarative and imperative in the system TAGGING); it is also possible to have conjunction of features, or any combination of disjunction and conjunction of features as an entry condition. The use of disjunctive entry conditions allows network to represent generalizations. For instance, the network represents the generalization that tagging a clause is an option if it is declarative (*you accepted, didn't you*) or imperative (*accept, will you*).

In generation, the grammatical system network is traversed system by system; a system is entered when its entry condition has been satisfied.

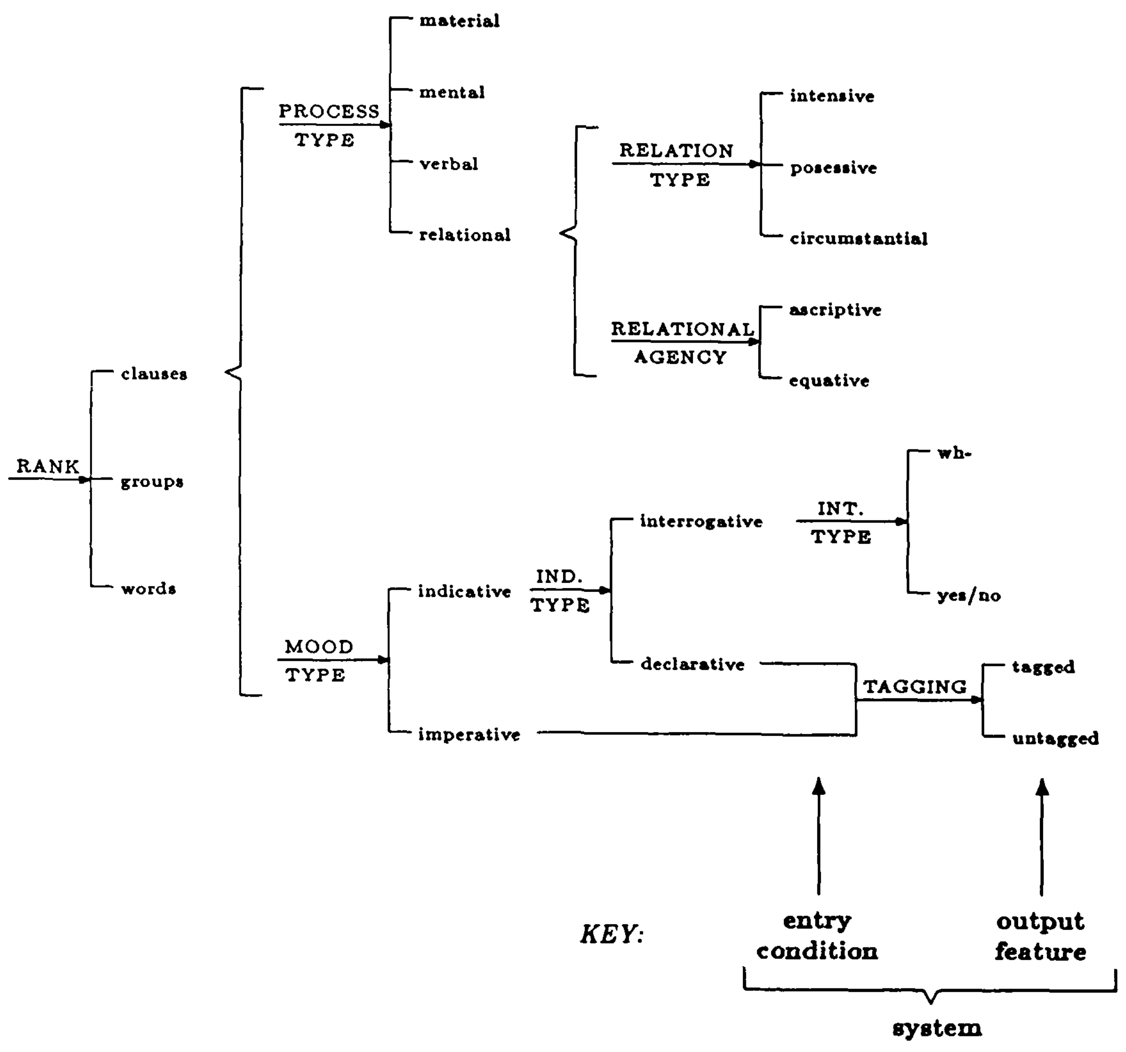


Figure 2-1: System network fragment

The system network is the locus of two additions, (i) "alongside": realization statements associated with features; and (ii) "above": choosers associated with systems.

2.2 Alongside the System Network: Realization Statements

Any feature may have one or more associated **realization statements**, written inside rounded-corner boxes in the diagram in Figure 2-2.[5] A realization statement specifies a structural fragment in the context of a feature of the system network.

For instance, the feature "indicative" has the realization statement "+Subject", which means *insert the grammatical function Subject* and ensures the presence of *Subjects* in indicative clauses. Other realization statements are also specified in this way in the context of system features.

There are a number of different types of realization statements: insertion, conflation, expansion, preselection, (lexical) classification, lexification, and ordering. We do not have to discuss the details of these different types, but it is important to note that each type specifies a single structural relation. In particular, the ordering of two functions, such as "Subject ^ Finite" (*order Subject before Finite*), is separated from the specification of their presence ("+Subject", *insert Subject*) and the two specifications can occur in different contexts. It is thus not necessary to specify the ordering of a function when it is introduced.

2.3 Above the System Network: Choosers And Inquiries

The system network and the realization statements associated with features in the network make up the grammar component of Nigel. The other component is the chooser and inquiry interface between the grammar and the environment (cf. Figure 1-1 above). The interface operates in terms of the systems of the system network. Each system in the system network has an associated **chooser** or choice expert. Choosers are drawn as boxes in the diagram in Figure 2-2; the associations to systems are represented by vertical lines. The diagram in Figure 2-2 is thus an enlargement of part of the Nigel box in Figure 1-1. The double line represents the boundary within Nigel between the chooser and inquiry interface and the systemic grammar.

A chooser is a semantic procedure which knows how to make a purposeful choice among the features of the system it is associated with. It makes the choice by asking one or more questions, called **inquiries**, thus obtaining the information relevant to the

[5]We have omitted the systems PROCESS TYPE, RELATION TYPE, and RELATIONAL AGENCY from Figure 2-1.

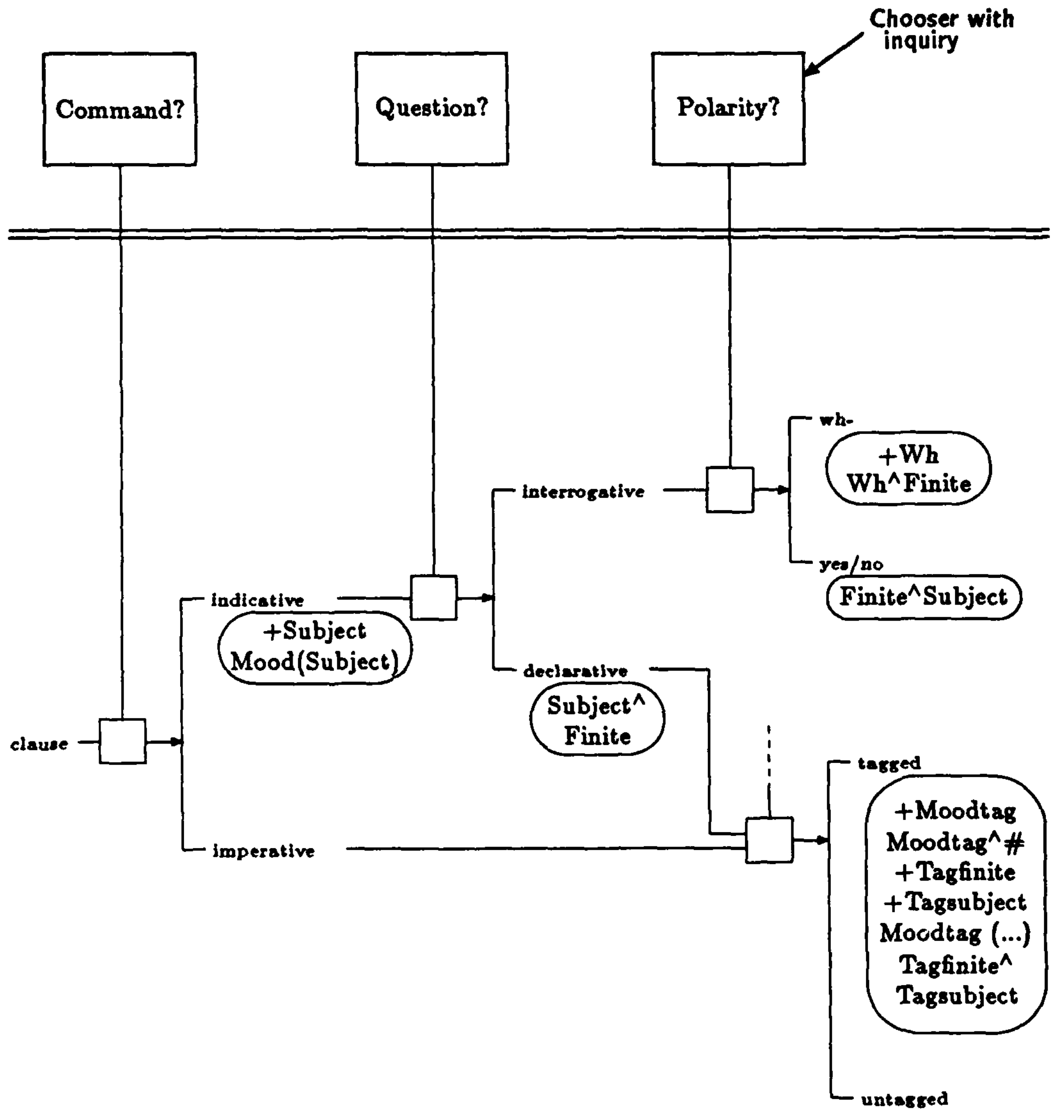


Figure 2-2: System network with choosers & realization statements

decision. We will give a more detailed account of the organization of a chooser presently.

2.4 Sequential Traversal of the Grammar

We have mentioned that the system network organizes grammatical generation: the basis of the generation is the traversal of the system network. We will now sketch a sequential algorithm for traversing the network. This traversal algorithm is part of the grammatical algorithm that controls the successive generation of grammatical units (such as clauses, nominal groups, and prepositional phrases). The grammatical generation algorithm only calls the traversal algorithm for one grammatical unit at a time.

In general, the traversal is from left to right in the network, entering a system when its entry condition has been satisfied. Since the systems are only partially ordered and the network has breadth as well as depth, it leaves the traversal order partly underdetermined. That is, when several systems can be entered, the network does not impose any restrictions on the traversal sequence, and the result is independent of the order in which the systems are selected and entered. The algorithm keeps a list of *enterable (waiting) system list*: a list of systems whose entry conditions have been satisfied and which are thus waiting to be entered. Initially, the enterable system list has only one element: the RANK system, whose entry condition is satisfied by the feature "start," which is the root of the system network. The traversal algorithm is represented diagrammatically in Figure 2-3.

The system network traversal algorithm:

Step 1: pick any (or the first) system from the enterable system list. The system picked becomes the current system.

Step 2: enter the current system. The chooser of the system is in charge of selection of features. The chooser is itself a decision tree with certain additional operations, which is traversed one step at a time until a choice has been made (see Section 2.5 below for more detail).

Step 3: once a feature has been selected, any associated realization statements are executed.

Step 4: update the enterable system list via an entry condition test. That is, add those systems whose entry conditions have been satisfied to the enterable system list.

Step 5: If the enterable system list is empty, exit. Otherwise go back to Step 1.

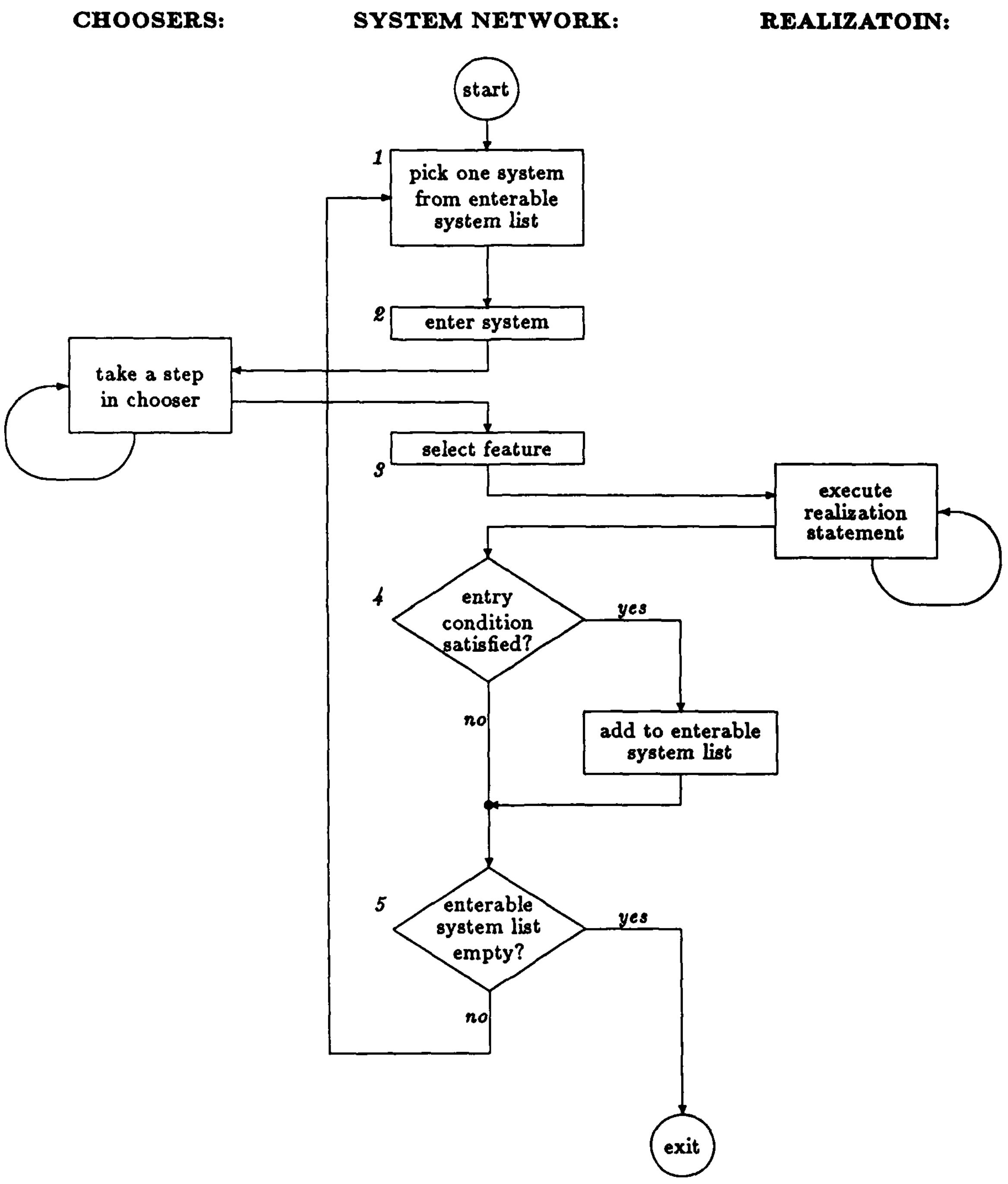

CHOOSERS:
SYSTEM NETWORK:
REALIZATOIN:
start
1
pick one system from enterable system list
2
enter system
take a step in chooser
select feature
3
execute realization statement
4
entry condition satisfied?
yes
no
add to enterable system list
5
enterable system list empty?
yes
no
exit

Figure 2-3: The sequential traversal algorithm

2.5 The Organization of a Chooser in More Detail

Whenever a new system is reached, its chooser is activated. The chooser presents its inquiries to Nigel's environment and chooses according to the responses; see Figure 2-4. Once the choice has been made and any realization statements associated with it have been executed, the traversal of the system network is resumed. This process continues until the grammatical unit to be generated has been fully specified. In general, the grammar is re-entered (and the system network is traversed) several times to generate a complete sentence.

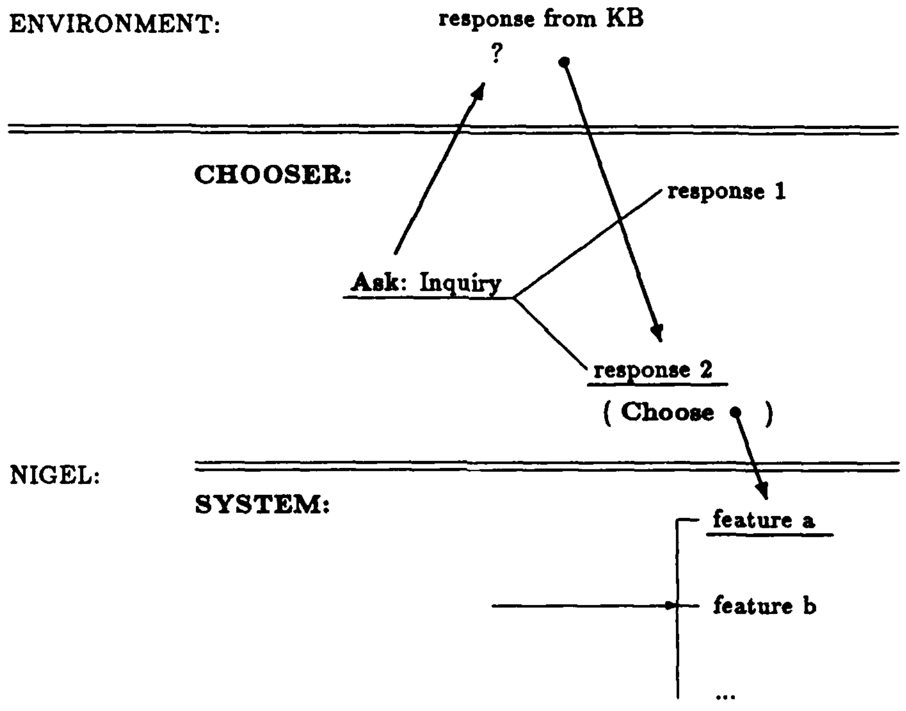


Figure 2-4: A system and its chooser in generation

A chooser consists of a number of chooser operations. Two of them introduce inquiries, viz. Ask and Identify. The others include CopyHub and Choose.[6] These four operations are described below:

[6]These four operations are the most important. We will not discuss two other operators, Pledge and TermPledge here.

1. **Ask** presents a branching inquiry to Nigel's environment. A branching inquiry has a fixed set of responses (usually two), one of which the environment has to return to the chooser. The overall organization of a chooser is the same as a decision (discrimination) tree, and each branching point in the tree is defined by Ask operation. For example, in the branching inquiry ModalityQ, either of the branches modal or nonmodal will be taken:

```
(Ask  (ModalityQ PROCESS)
  \ \
   \  \(modal  (Identify FINITE (ModalityID PROCESS))
    \           (CopyHub (SPEAKINGTIME TEMPO1))
     \          (Choose Modal))
      \
       \(nonmodal  (CopyHub SPEAKINGTIME TEMPO0)
                    (Choose Temporal) ))
```

2. **Identify** presents the identifying inquiry to the environment and associates the response with the grammatical function (such as SUBJECT, ACTOR, and POLARITY). The set of responses is open-ended. The resulting association pair is recorded in the **Function Association Table** (FAT):[7]

Hubs[8]	**Grammatical Functions**
WREN-GAZEBO-STATEMENT	SPEECH-ACT
NOW	SPEAKING-TIME
GAZEBO-BUILDING	PROCESS
HISTORIC-TIME	EVENT-TIME

If a grammatical function appears in the table at a given point in the generation process, an association has already been produced for it. Otherwise, there is no such association at that time and it has to be produced if the function is to be used as a parameter in an inquiry.
Identify takes a grammatical function and an identifying inquiry as operands, e.g. (Identify FINITE (ModalityID PROCESS)).

[7]Example from [MannMatthiessen 83].

[8]A hub is a conceptual locus in the environment. A hub associated with a function is typically a concept or a constant from the knowledge base.

3. **CopyHub** is simply used to copy an existing hub association from one grammatical function to another, e.g. (CopyHub SPEAKINGTIME TEMPO0).
4. **Choose** takes a grammatical feature as its operand (e.g. (Choose Modal)); it is the chooser operator that actually chooses one of the feature options in the chooser of a system. Each terminal branch in a chooser tree must have a Choose operator associated with it.

A series of these operations enable each system to present inquiries to the environment, to process the answers, and to choose features. The selected features are then used by Nigel to construct clause structure, and items on the Function Association Table are used to place selected words correctly.

3. Parallelism in the Nigel Component

In the Nigel grammar and in systemic grammars in general, the system is the representation of a grammatical choice point and its related operations, which makes it a natural logical unit. The system is also a computation unit, since the traversal algorithm (see Section 2.4) repeats the cycle of entering-choosing-realizing once for each system. It is therefore natural to choose the system as the building block of our parallel computational model.

3.1 A Parallel Computation Model

In our parallel computation model, each system and its associated chooser and realization statements are viewed as a computation unit -- what we will call a system process. It is shown in Figure 3-1.

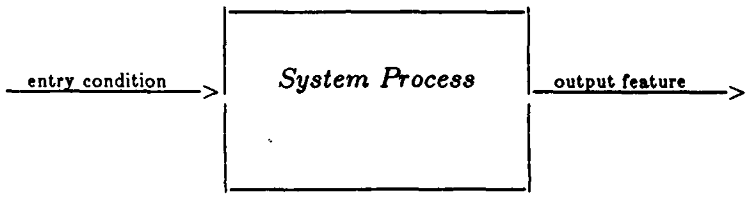


Figure 3-1: The building block in the parallel process model

As we have seen, the only necessary condition for a system to be enterable is the satisfaction of its entry condition. It is also the only necessary condition to activate a system process. However, certain systems may occasionally be "delayed" by its associated chooser operations -- a chooser operation in such system may not start before the creation of a certain entry in the Function Association Table (FAT).

These two are the only conditions on the simultaneous computation of all systems that have been observed. They give rise to dependencies among systems and choosers:

1. **System dependency** -- a system cannot be activated before its entry condition is satisfied. On the other hand, a system can be activated as soon as its entry condition satisfied.
2. **Chooser dependency** -- each inquiry of a chooser depends on one or more item in the Function Association Table, and the table is filled by all choosers.

Thus, there is a producer-consumer relation between one chooser and another. A consumer has to wait for the producer to generate a certain item if it is not already in the table.

While the first dependency is obvious, the second is not. Fortunately, in most cases, the chooser dependency is weaker than the system dependency. That is, the partial ordering defined by the former could be subsumed by the partial ordering defined by the latter. In a few cases, however, there is chooser dependency even when there is no system dependency: systems that are not ordered with respect to one another but are on simultaneous paths in the system network may have choosers in a dependency relation. In these cases, the dependency can be handled in two alternative ways:

1. a consumer system process must wait for all producer systems to finish their tasks; or,

2. if finer-grained parallelism is assumed,[9] the consumer and producer systems can be activated in an overlapped fashion, and only the chooser operation in the consumer need to wait for the chooser operation in the producer to finish.

The first implementation defines an enterable system as a system satisfying both kinds of dependency. In the second implementation, an enterable system is a system satisfying system dependency, but a synchronization point must be inserted into those consumer chooser operations where the chooser dependency should prevail. One of these few cases will be illustrated in an example in Section 3.3.

3.2 Four Parallel Schemes

The above computation model combines with parallelism constraints to provide a global picture of parallel processing in Nigel. This section sketches four possible ways of parallelizing the generation algorithm for traversing the system network discussed above:

1. Within the system network itself: simultaneous systems can be processed in parallel.
2. Across choosers, systems, and realization statements: these different 'levels' of abstraction can be processed in parallel the opportunity arises.
3. Within a chooser: certain chooser operations can be processed in parallel.
4. Within a system: realization statements can be processed in parallel.

We will now look at these in some more detail and then illustrate them in the generation of a clause.

[9]That is, if we assume that the chooser operation, rather than the system, is the computation unit.

3.2.1 In system network: simultaneous systems

In the first step of the algorithm, pick *all systems from the enterable system list* instead of just one. Alternatively, let's use our parallel model and make the following assumptions. (i) Every system process has its own processing power, then all systems may check their entry conditions constantly and those that are enterable become active. Each such system will execute Step 2 through 4 of the algorithm independently. (ii) The algorithm terminates when the enterable system list is empty and no system process is active.

Since a system can be entered when its entry condition is satisfied, all systems on multiple paths can be entered simultaneously as soon as their corresponding entry conditions are satisfied. The traversal of the system network is thus parallel breadth-first traversal.

3.2.2 Simultaneous systems, choosers, and realization statements

The computation of systems, choosers and realization statements can be parallelized. That is, after a feature is selected by the chooser, a new system which can then be entered may start to work while the chooser continues (to add information to FAT). Besides, the realization tasks in the original system can also be started at that time.

These correspond to starting Step 3 and 4 above in the middle (or at the end) of Step 2: at the point where the output feature is selected.

3.2.3 In chooser: simultaneous chooser operations

The operations of a chooser can be parallelized. If a chooser makes two inquiries that are independent of each other, they can be processed in parallel and consequently save some time in selecting a feature.

This corresponds to parallel processing within Step 2 of the traversal algorithm.

3.2.4 In system: simultaneous realization statements

The execution of a realization statement is conditioned by the feature it is associated with: When the grammatical feature is chosen, any realization statements associated with it can be executed. It is important to note that the logic of grammatical choice and the structural consequences of particular choices are *factored into independent representations* in the grammar framework, i.e., into the system network and into realization statements associated with particular feature options in the system network. This means that the traversal of the system network is logically independent of the execution of realization statements.[10] This corresponds to parallel processing within Step 3 in the traversal algorithm.

Next, we illustrate the opportunities for parallelism and its restriction in Nigel by a real, typical example.

3.3 An Example of opportunities for parallelism

This example is the first pass (first traversal of the system network) of the Nigel generation process that generates a sentence "the new system is more reliable than the old one" (Figures 3-2a, b and c).

In these figures, each box represents a system and its associated chooser. Each box is pointed to by a feature that satisfies the entry condition of the system, and points to a feature the chooser selects. The symbol "#" represents the termination of a branch. A branch is terminated when a previously selected feature does not satisfy the entry condition of any system. The number within each box denotes the number of realization statements in that system.

The system "LexicalVerbTermResolution" is not shown in these figures, but it is the last thing to be executed in the first pass of the generation process.

All four kinds of parallelism can be found in this example. For instance, (1). the first kind of parallelism can be found in Figure 3-2a, where eight new systems triggered by "TransitivityUnit,"

AccompanimentAdjunct,
CauseAdjunct,
ExtentAdjunct,
Location,
MannerAdjunct,
MatterAdjunct,
ProcessType,
RoleAdjunct,

can all be executed in parallel. This kind of parallelism is also found in many places in Figures 3-2b and 3-2c.

(2). The second kind of parallelism is found in Figures 3-2b and 3-3: when feature

[10]*In this respect, the grammar differs fundamentally from other frameworks where grammatical choice and structure building are collapsed into the same rule type and are thus logically inseparable.*

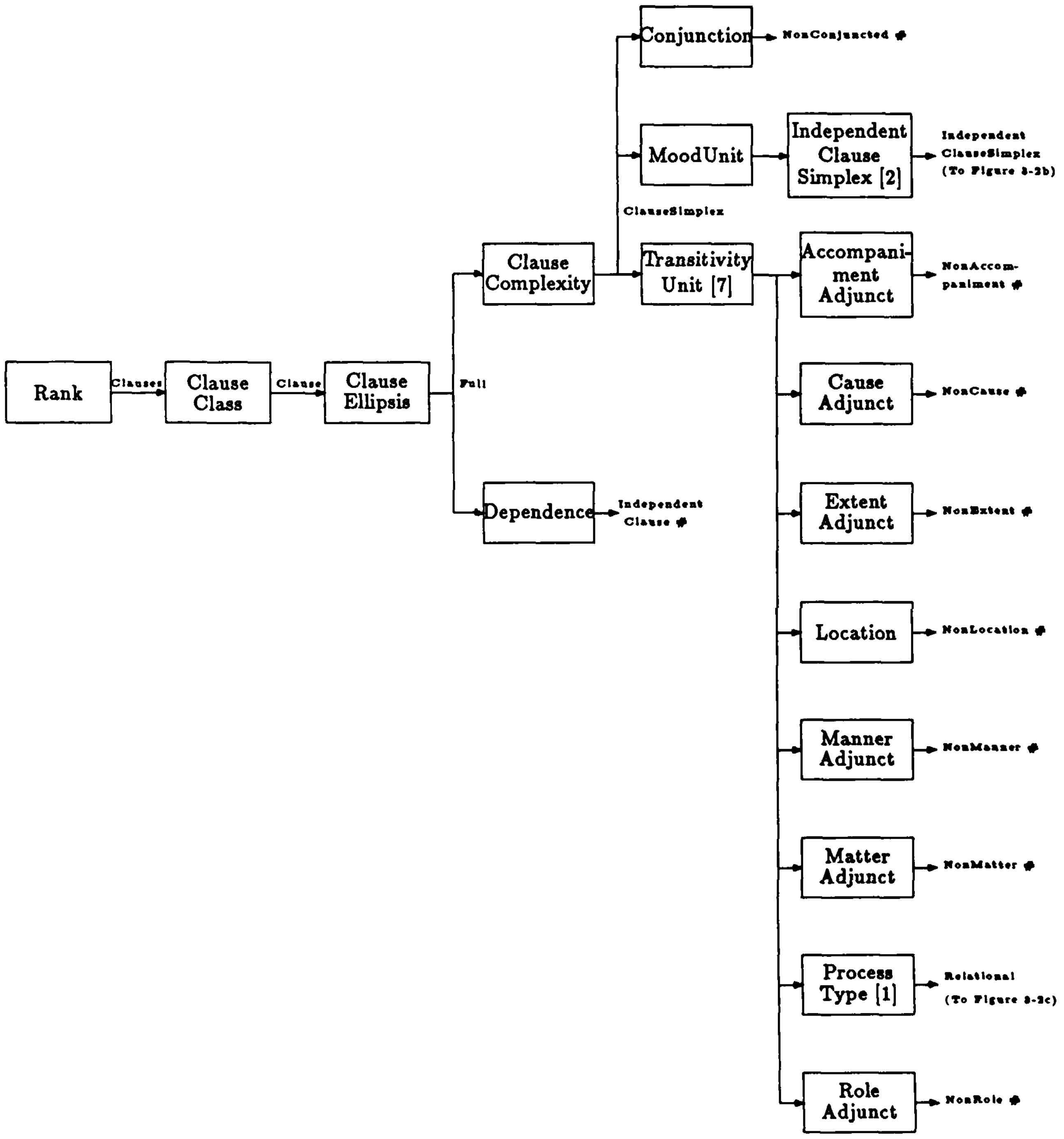


Figure 3-2a: System network of the example

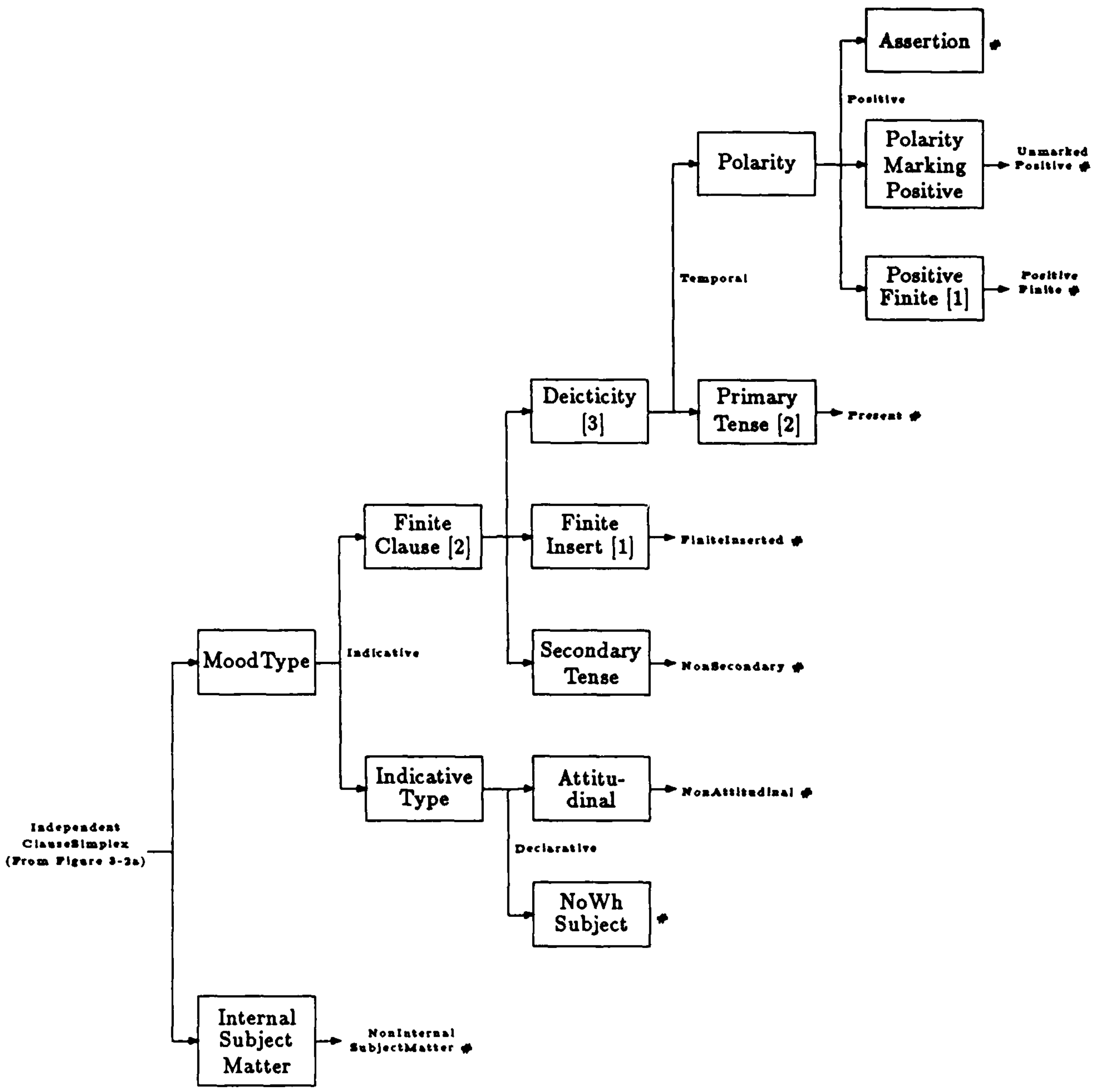


Figure 3-2b: System network of the example (cont.)

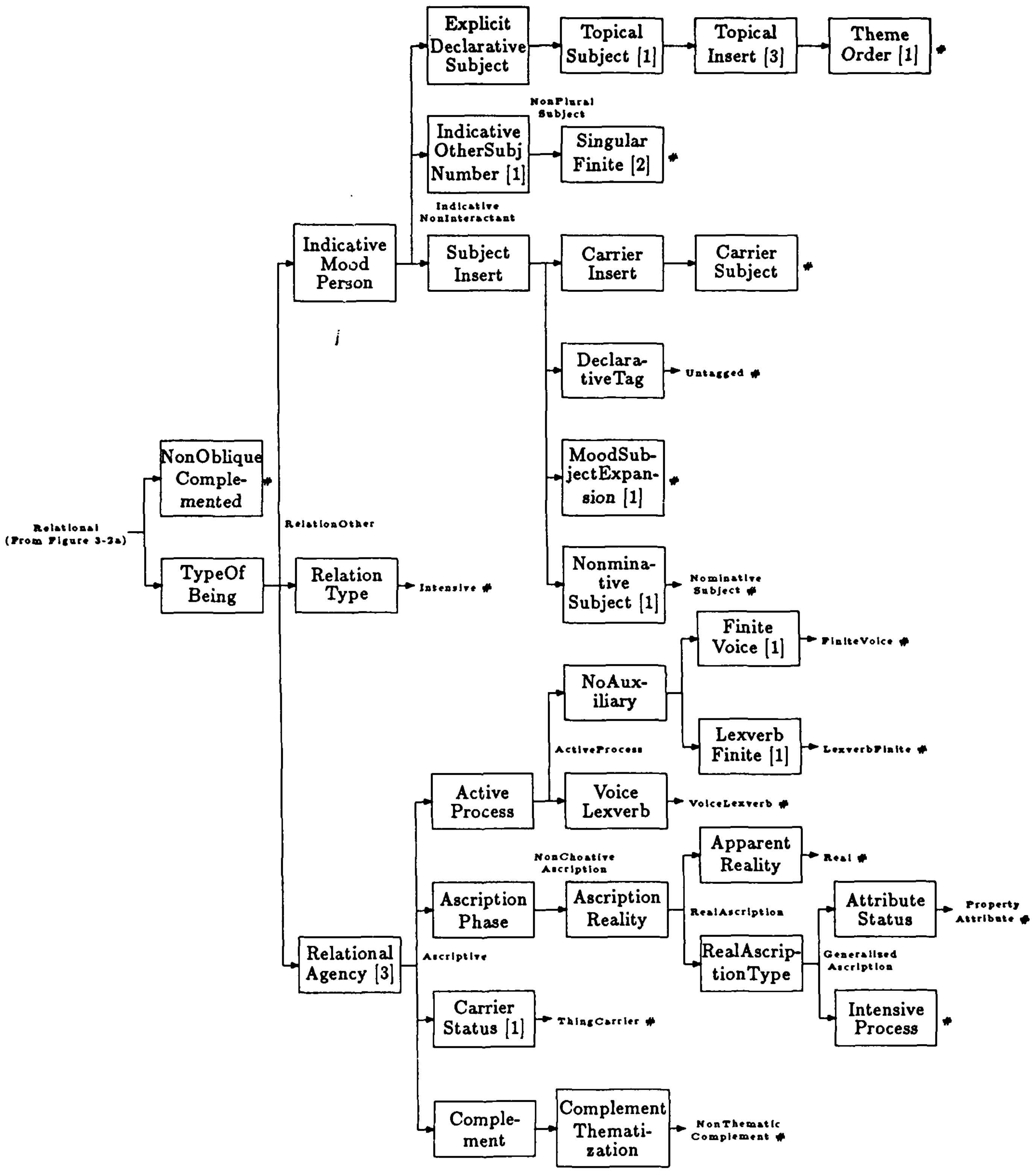


Figure 3-2c: System network of the example (cont.)

"Temporal" is selected by system "Deicticity," two new systems "Polarity" and "PrimaryTense," three realization tasks in Deicticity, and the remaining work of Deicticity chooser (adding information to the FAT) can all be executed in parallel.

(3). The third kind of parallelism exists in the chooser "PrimaryTense" (Figure 3-3) where its operation 5 asks if TEMPO1 precedes TEMPO0, and operation 6 asks if TEMPO0 precedes TEMPO1. These two "ask" operations can be processed in parallel.

(4). The fourth kind of parallelism can be found in the same figure where the three realization tasks in system Deicticity:

(Insert TEMPO0),
(Conflate TEMPO0 FINITE) and
(OutClassify FINITE Reduced)

can be processed in parallel.

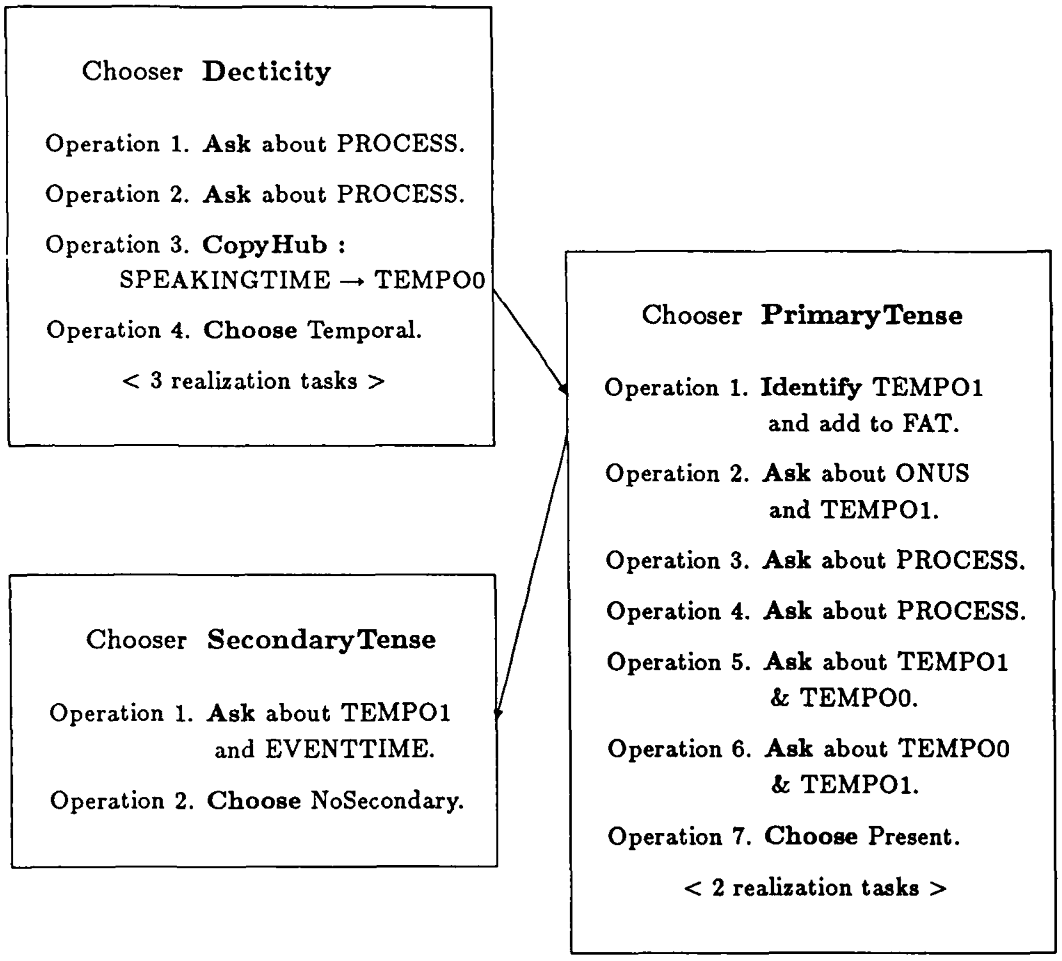
While systems in the same branch are able to start simultaneously, their associated choosers are not necessarily so due to the chooser dependency explained earlier. There are a few cases where the chooser dependency dominates. For example, in Figure 3-2b, systems "Deicticity" and "SecondaryTense" can start simultaneously but their choosers cannot, because

1. The Deicticity chooser creates an association on FAT for the grammatical function "TEMPO0";
2. The PrimaryTense chooser uses TEMPO0 and creates an association for "TEMPO1" in its first identify operation;
3. The SecondaryTense chooser uses TEMPO1 in its first ask operation.

So the resulting execution ordering should be : 1. Deicticity, 2. PrimaryTense, then 3. SecondaryTense. In other words, the SecondaryTense chooser cannot start its work in parallel with Deicticity chooser as implied by the system network.

The detailed operations performed by these choosers and the chooser dependency relations are shown in Figure 3-3. Note that the execution ordering (Deicticity, PrimaryTense, SecondaryTense) only defines the starting sequence of these choosers. They can be executed in an overlapped way: After the third operation of Deicticity "CopyHub: SPEAKINGTIME → TEMPO0," PrimaryTense can start its first operation "Identify," which uses TEMPO0. And right after PrimaryTense has created an association for TEMPO1, SecondaryTense can start its first operation "Ask," which uses TEMPO1.

Since SecondaryTense has only two operations, it could finish even before the completion of PrimaryTense. So parallelism can still be useful even in this case.



Chooser **Decticity**
Operation 1. **Ask** about PROCESS.
Operation 2. **Ask** about PROCESS.
Operation 3. **CopyHub** : SPEAKINGTIME → TEMPO0
Operation 4. **Choose** Temporal.
< 3 realization tasks >
Chooser **PrimaryTense**
Operation 1. **Identify** TEMPO1 and add to FAT.
Operation 2. **Ask** about ONUS and TEMPO1.
Operation 3. **Ask** about PROCESS.
Operation 4. **Ask** about PROCESS.
Operation 5. **Ask** about TEMPO1 & TEMPO0.
Operation 6. **Ask** about TEMPO0 & TEMPO1.
Operation 7. **Choose** Present.
< 2 realization tasks >
Chooser **SecondaryTense**
Operation 1. **Ask** about TEMPO1 and EVENTTIME.
Operation 2. **Choose** NoSecondary.

Figure 3-3: Chooser operations and chooser dependency

3.4 Analysis of the Four Parallel Schemes

The first parallel scheme is the major source of parallelism in Nigel. It is a parallel breadth-first traversal of the system network that allows several systems to be started simultaneously if their entry conditions are satisfied and no chooser dependency exists between any two of them. When this scheme is exploited, the time required for each traverse of a system network is only the time spent on the longest path in that network, instead of the total computation time of all systems as in the sequential processing case. If we assume that

1. the network (for given traverse) has n systems $s_1, s_2, \ldots$ through s_n;
2. each system s_i takes t_i time units;
3. the longest path is composed of k systems $s'_1, s'_2, \ldots$ through s'_k, where $s'_1 = s_1$ and each of other s'_i is some s_i; each s'_i takes t'_i time units; and
4. inquiries are always immediately serviced by the Nigel environment;

then the speed-up factor for each traverse using this parallel scheme is $\frac{\sum_1^n t_i}{\sum_1^k t'_i}$.

If we further assume that all systems take the same amount of time, then this factor is simplified to $\frac{n}{k}$, or the average branching factor. In the example in the previous subsection, the longest path has 12 systems while there are 62 systems in total.

The second parallel scheme means that the system network can be contracted in such a way that some systems are overlapped with others at the points of their choose-feature operations. Thus if a chooser has something else to do after selecting a feature, or if there are some realization statements in the system, these operations can be performed simultaneously with another system. The benefit from this scheme depends on how early a chooser can select its feature in its lifetime, and how much computation the realization statements require. In practice, most systems spend the most time in selecting a feature, so the speed-up benefit from this parallel scheme depends on the realization statements. This is not likely to be very large.

The third parallel scheme assumes that the height of the decision (discrimination) tree formed by "ask" operations can sometimes be reduced. Consider a complete binary decision tree in which each node (except a leave) is a decision point, and whose height is h. It would require h "ask" operations to traverse the tree sequentially. If all decision nodes are processed simultaneously, the traversal would only take as long as the longest ask operation in the tree. The resulting speed-up factor is h, assuming each "ask" operation takes equal amounts of time. The drawback of this scheme is that it needs to process 2^h-1 nodes at a time, versus one node at a time in the case of sequential processing. Since this would require considerable computing resources when h is large, we may assume that the speed-up h would not be very large in practice.

The fourth parallel scheme is straightforward; it always exists whenever the number of realization statements in the system is larger than one. However, its effect could be masked by the first two parallel schemes and is almost insignificant when both these schemes are employed.

Based on what we have noted, a reasonable estimate of the potential speed-up benefit from these four parallel schemes is between one to two orders of magnitude. This figure can be more significant when combined with a faster Nigel environment or when a multiple-Penman system is employed to generate more than one sentence at a time.

3.5 A Linguistic Functional View of the Parallelism Study

Parallel processing has benefits over sequential processing beyond the promise of increased computing speed. It offers a more natural framework for modelling the linguistic theory underlying Nigel.

The theory, now called systemic (functional) theory, was originally formulated by Michael Halliday (1961). He has developed the theory in such a way that it represents linguistic activity or language processing as a set of differentiated "subprocesses" which can work in parallel. In a sense, the theory serves as a prism that breaks up language processing into simultaneous parts. The following have been central to our discussion.

(i) Stratal differentiation: in speaking and writing, we are engaging in both semantic processing and grammatical processing and they constitute two different levels or strata of processing rather being sequential stages in the application of the same rule system.

(ii) Differentiation into choice and the structural realization of choices made. In systemic theory, grammar is organized as a network of inter-related choices between grammatical alternatives. The process of choosing in this network is separated from the process of realizing particular choices by specifying fragments of grammatical structure. In contrast, the two processes are collapsed into one formalism in other grammars such as phrase structure grammars and transition network grammars.

(iii) Metafunctional differentiation: in speaking and writing, we do three things at once. We represent some part of our experience, we interact with a listener or reader, and we present the representation and interaction as text in context. To enable us to do this, the grammar is organized into three simultaneous metafunctional components; the ideational metafunction for representation, the ideational metafunction for interaction, and the textual metafunction for the creation of text; they constitute simultaneous choices available to the speaker or writer.

These differentiations have often either been obscured, missed entirely or else represented sequentially in other grammatical frameworks. For example, Halliday (1978: 134) comments on the metafunctional organization.

> ... a linguistic system is not a progressive specification of a set of structures one after the other, ideational, then interpersonal, then textual. The system does not first generate a representation of reality, then encode it as a speech act, and finally recode it as a text, as some writing of philosophical linguistics seems to imply. It embodies all these types of meaning in simultaneous networks of options, from each of which derive structures that are mapped onto one another in the course of their lexicogrammatical realization. The lexicogrammar acts as an integrator, taking configurations from all components of the semantics and combining them to form multilayered, 'polyphonic' structural compositions.

Our general point is that while Halliday's framework works well in a sequential implementation, its design characteristics make it an important candidate for a parallel implementation. Simultaneity is emphasized and all sequential orderings are intrinsic to the organization of choice in the grammar itself. There are no extrinsic rule orderings created artificially because of the nature of the grammatical formalism. Moreover, the framework can be fully realized only in a parallel implementation.

4. The KL-TWO Component

The role of the knowledge base in Penman computation is to recognize each inquiry operator in Nigel and to respond to each one appropriately [Mann 83b]. In this section, we shall explain how KL-TWO works as an independent knowledge base, and how it is used as the knowledge base in the Penman system.

KL-TWO is a hybrid knowledge representation system linking two reasoning systems: NIKL and PENNI. NIKL is a terminological reasoner in which one can define universally quantified sentences. On the other hand, PENNI is an assertional reasoner containing a database of propositional assertions, it does not contain any quantification and is simple. The resulting system (i.e. KL-TWO) does not have the full representational expressiveness of a first-order language, but it has higher computation efficiency instead [Vilain 85]. The structure of the entire KL-TWO system is shown in Figure 4-1, where PENNI has two subcomponents: RUP and an interface to NIKL.

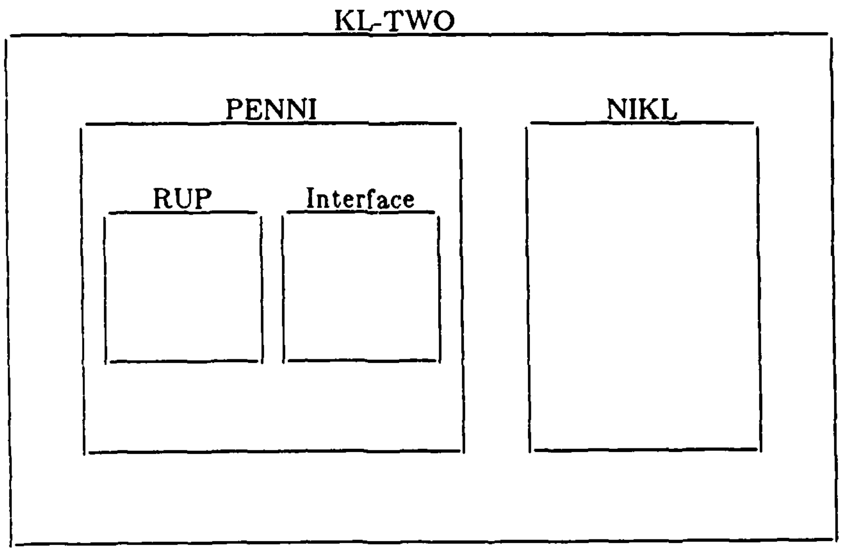


Figure 4-1: The organization of the KL-TWO

4.1 NIKL

NIKL is a network knowledge-base system descended from KL-ONE [BrachmanSchmolze 85]. This type of reasoner supports description of the categories of objects, actions, and states of affairs that make up a domain. The central components of the notation are sets of concepts and roles, which are organized in IS-A hierarchies. The concepts are used to identify the categories of entities. The roles are associated with concepts (as "role restrictions"), and identify the relationships that can hold between actual individuals belonging to the categories. Number restrictions on these role restrictions identify how many individuals can fill the given relationship. Primitive concepts identify "natural kinds," i.e., classes whose membership cannot be established by definition. The IS-A hierarchies identify when membership in one category (or the holding of one relationship) entails membership in (or the holding of) another.

NIKL is a basically a frame system in which the concepts are equivalent to frames and the role restrictions are equivalent to slots. However, the NIKL representation can be given a formal semantics. Because of this NIKL can support a special kind of reasoner, called a "classifier", that can compute "subsumption" relations. So a concept **A** subsumes a concept **B** only if the set denoted by **A** includes the set denoted by **B**. The classifier "classifies" a new concept in a NIKL domain model *below* all and only all concepts which subsume it and *above* all and only all concepts that it subsumes.

The algorithm for subsumption can be found in [SchmolzeLipkis 83]:

Input: Two concepts (or roles) **A** and **B**;

Output: Truth of (**A** subsumes **B**);

The algorithm returns true if both:

1. all primitive concepts that subsume **A** also subsume **B**; and
2. for each role restriction of **A**, some role restriction of **B** denotes the same relationship, and for those corresponding role restrictions:

   (i). The number restriction for **A**'s role restriction includes that of **B**'s.

   (ii). The value restriction of **A**'s role restriction subsumes that of **B**'s.

Let's consider an example of testing the truth value of "**Parent** subsumes **Grandparent**":

- **Parent** is a non-primitive concept, is subsumed by the primitive concept **Person**, and has a role restriction **Child** with a number restriction of one or more, and a value restriction of **Person**.
- **Grandparent** is a non-primitive concept, is subsumed by the primitive

concept **Person**, and has a role restriction **Child** with a number restriction of one or more, and a value restriction of **Parent**.

Parent and **Grandparent** have the same primitive subsumers, namely, **Person**. Looking at the **Child** role restriction, they have the same number restriction at each concept. Finally, looking at the value restriction of **Child** at **Parent**, we can see that it subsumes the value restriction of **Child** at **Grandparent**. Hence **Parent** subsumes **Grandparent**.

4.2 PENNI

PENNI is an enhanced version of McAllester's RUP [McAllester 82]. It includes RUP and an interface to NIKL.

4.2.1 RUP -- PENNI's major component

RUP (Reasoning Utility Package) is a propositional reasoner developed by McAllester [McAllester 82]. It contains a truth maintenance system (TMS) which can be used to perform simple propositional deduction, and an equality system which handles substitution of equals for equals. RUP also has a demon-like facility capable of some quantificational reasoning, which does not come into play in the Penman system.

The major operation of importance to Penman performed in RUP is simple deduction. These deductions are comparatively simple because they do not involve quantifiers. A single deduction operation is similar to a database retrieval operation. For example with the fact **(PERSON John)**, the query **(PERSON ?X)** will deduce **"X is John"**. Assertions can use the full power of Boolean logic as can patterns given to the inference mechanism.

4.2.2 PENNI interface -- interfacing RUP and NIKL

For the sake of the interface, PENNI can be viewed as managing a database of propositions of the form **(P a)** and **(Q a b)** where the forms are variable free. The first item in each ordered pair is the name of a concept in an associated NIKL network, and the first item in each ordered triple is the name of a role in that network. So the assertion of any form **(P a)** is a statement that the individual **a** is a kind of thing described by the concept **P**. Furthermore, the assertion **(Q a b)** states that the individuals **a** and **b** are related by the abstract relation described by **Q**. For example, **(Man Don)** and **(Child John Mary)** are valid PENNI expressions.

Via the interface, NIKL adds to RUP the ability to perform classification reasoning [Vilain 85].

4.3 Using KL-TWO as an External Environment for NIGEL

As has been stated, logical forms are submitted as a demand for expression to Penman. These forms must be interpreted by the Nigel inquiries. Nigel must be able to decompose the expressions and characterize their parts. To achieve this, NIKL is used to categorize the concepts (or terms) of the domain in terms of Nigel's implicit categorizations. Nigel inquiries have been implemented which use the structure of the logical language and a NIKL model to analyze the logical forms. To do this efficiently, the logical form have been translated into a KL-TWO database. This works by translating the logical forms into two separate structures that are stored in a RUP database. A logical form is translated into one structure that shows the connection of terms in the formula. All predications appearing in the logical form are put into the RUP database as assertions.

Figure 4-2 shows the set of assertions entered for the formula in Figure 4-3. A second structure shows the quantifier scopings through a tree.

```
(ActionOccurrence x)
(Past p)
(timeofoccurrence x p)
(Transmit y)
(records x y)
(actor y Smith)
(Message z)
(actee y z)
```

Figure 4-2: Sample PENNI assertions

The implementation of Nigel's inquiries using the connection and scope structures with the NIKL model is fairly straightforward to describe. Since the logical forms reflecting the world view are in the NIKL model, the information decomposition inquiries use these structures to do search and retrieval. With all of the predicates in the domain specializing concepts in the functional systemic level of the NIKL model, information characterization inquiries that consider aspects of the connection structure can test for the truth of appropriate RUP propositions. The inquiries that relate to information

$$\begin{array}{l}(\exists\, x \in \text{ActionOccurrence})\ ((\exists\, p \in \text{Past})\ (\text{timeofoccurrence}(x,p))\wedge \\ \quad (\exists\, y \in \text{Transmit})\ (\text{records}(x,y)\wedge \text{actor}(y,\text{SMITH})\wedge \\ \qquad\quad (\exists\, z \in \text{Message})\text{actee}(y,z)))\end{array}$$

Figure 4-3: Example logical expressions

presented in the quantification structure of the logical form will search the scope tree. To supply lexical entries, lexical entries are associated with NIKL concepts as attached data and use the retrieval methods of RUP and NIKL to retrieve the appropriate terms. Finally, discourse information is also stored in and retrieved from the RUP database. Examples of this process can be found in [SondheimerNebel 86].

5. Parallelism in KL-TWO

The performance of KL-TWO affects that of Penman directly. In the current implementation of Penman, we conjecture that KL-TWO takes at least as much time as (if not substantially more than) Nigel when they are used to generate a sentence. Since the parallel processing of Nigel also means that more inquiries are presented to KL-TWO in unit time, a speeding up of KL-TWO seems to be necessary for the whole Penman system to gain any benefit from parallel processing. In Section 1, we suggested an obvious solution: to use as many KL-TWO systems as are needed to match the number of simultaneous inquiries. However, this solution requires multiple computing resources, so it is very expensive. Besides this solution does not help to balance KL-TWO with a sequential Nigel. To overcome these problems, an alternative solution can be used to replace, or enhance, the first solution. That solution is to exploit parallelism within KL-TWO itself.

We sketch the idea of parallelizing operations in KL-TWO below. We then survey two existing approaches which have realized this idea: Forrest's Classifier System approach [Forrest 85, Forrest 87], and Derthick's Connectionist Models approach [Derthick 86].

5.1 Parallelizing NIKL and PENNI

In a frame-based language such as NIKL, Classification (subsumption) is the most important computation. So it is interesting to find out whether one can apply parallel computation to Classification. The answer is both yes and no. No, parallel processing cannot help much in this case, since it has been proved that the classification problem of a frame-based description language with the expressiveness of role restriction is a *co-NP-complete* problem [BrachmanLevesque 84]. As a result, the classification computation in NIKL is very likely to be inefficient by its nature. But yes, we may get around the problem by not attempting completeness, as in the case of the current implementation of NIKL [KaczmarekBatesRobins 86], since we might not always care to compute all possible solutions. Further improvement by using parallel processing has also been demonstrated to be appropriate [Forrest 85].

The basic idea of parallelizing the classification resembles that of parallelizing the system network in Nigel: we could use simultaneous breadth-first search to manipulate the NIKL network, so that the total computation time is proportional to the depth, instead of on the size, of the network.

In PENNI itself, the deduction operation is straightforward and can be implemented by any parallel database search scheme. For example, given a simple form, (**P** a), as a query, all simple forms with **P** in the first of the two positions can be matched simultaneously. Or if the query is in the disjunctive normal form, each disjunct can be matched simultaneously.

The above idea has been followed by Forrest [Forrest 85, Forrest 87] who used Classifier Systems to model NIKL and its operations,[11] and by Derthick [Derthick 86] who used Connectionist Models to model the whole KL-TWO. Their works are interesting in that both Classifier Systems [Holland 86] and Connectionist Models [Hinton 81, FahlmanHinton 87] are representative knowledge-representation and inference schemes with parallelism "built-in" their schemes at subcognitive level, and these schemes have been drawing broad attention in both the cognitive science and artificial intelligence communities. Although these works treat KL-TWO in a general view rather than treat KL-TWO as Nigel's environment, the results should still be valid since the Nigel inquiries are general enough to require most (if not all) KL-TWO operations, including classification.

5.2 Parallel NIKL Using The Classifier Systems Approach

The Classifier System is based on a standard production system. Knowledge in Classifier Systems is represented by condition-action rules, called classifiers. But unlike a standard production system, a Classifier System allows simultaneous firing of multiple rules. This parallelism solves not only the efficiency problem, but also the partial matching requirement by allowing the coexistence of a set of rules with different degree of speciality.

A Classifier System contains a set of classifiers each of which can perform one action, that of adding a message to the short term memory, called the message list. At any instant, the state of the message list determines which classifiers are eligible to write information to the message list at the next time step. In Classifier Systems, parallelism is exhibited at the following levels:

1. all positions in a condition are matched against all bits in a message simultaneously,
2. one condition is matched against all messages on the message list simultaneously,

[11] Forrest used the term KL-ONE in her work as a general term indicating the evolving language including NIKL, or a New Implementation of KL-ONE.

3. all conditions on one classifier are matched simultaneously,
4. all classifiers in a Classifier System are matched simultaneously, and
5. all classifiers are allowed to post their output messages to the message list simultaneously.

5.2.1 Forrest's work

Forrest used the Classifier Systems as the framework for parallel NIKL implementation. One classifier is assigned to represent every directed link in the NIKL network [Forrest 87]. Classifiers are coded in bit strings, and either "feature detection" or "tagging" can be used to implement inheritance structures -- the first technique uses don't-cares (#) to represent a concept's generality, and the second technique represents the inheritance links explicitly as classifiers. The resulting system solves the classification problem by parallel graph search[12] and fast boolean set operations.

The performance benefit of a parallel system using Classifier Systems depends on some factors characterizing NIKL's network been traversed. These factors include E, F, F_{eff} and D_{mid} which are defined below:

(i). E is the average fan-out per concept, that is, the mean number of subconcepts for every concept in the NIKL network;

(ii). F is the average fan-in per concept, that is, the mean number of superconcepts for every concept in the NIKL network;

(iii). F_{eff} is the effective fan-in meaning the difference between F and E in a certain region of the network; and

(iv). D_{mid} is some depth, above which F_{eff} is close to 1 and below which F_{eff} is significant.

Based on these assumption, Forrest was able to express the computation time for parallel and sequential classification algorithms by $O(D_{mid})$ and $O(F_{eff}^{D_{mid}})$, respectively [Forrest 85]. In other words, parallel processing may, under certain conditions, improve the efficiency of classification computation significantly.

[12]By parallel graph search is meant that, for example, when the value restriction of role R for a concept C is to be found, the algorithm first traverse up ISA links to find the set of superconcepts, then traverse all R links in parallel that originate in one of the superconcepts, and so on. This is done implicitly by the Classifier System mechanism.

5.3 Parallel KL-TWO Using The Connectionist Models Approach

Connectionist Models represent knowledge by means of activation patterns defined over simple nodes in a highly interconnected network. Each individual node has little information by itself. Information is conveyed by the statistical properties of pattern of interconnections and the *weight* (or *strength*) associated with each connection in the network. In other words, the knowledge is in the connections. By building a simple working model for semantic networks using this approach, Hinton was able to show that this idea is feasible [Hinton 81]. Based on this, Derthick [Derthick 86] proposed to develop a knowledge representation system that is functionally similar to KL-TWO, but implemented on a Boltzmann Machine network.

5.3.1 Derthick's work

The Boltzmann Machine is a massively parallel system operates in radically different way from logic circuits. In the Boltzmann Machine, each processing unit is very simple and is always in one of two states (0 or 1). The current state of any given unit is a probabilistic function of the state of neighboring units and a coefficient (the *weight*) indicating, for each pair of units, whether these units excite or they inhibit each other, and how the excitation / inhibition is. A subset of the units of a Boltzmann Machine are designated as *visible*. Problem *specifications* are given in terms of the states of these units. The rest of the units, designated *hidden* units, are available for use as desired to aid in properly constraining the behavior of the visible units.

Using a language μKLONE, which is similar to KL-TWO, concepts, roles, and assertions can be encoded in the Boltzmann Machine in the form of patterns of activity of visible processing units. Information is obtained from the machine by clamping a subset of the visible units, and allowing the rest of the units to asynchronously update their states. After a while, the unclamped visible units are examined, and their state represents the answer to the query.

By choosing patterns carefully, this machine can be used to answer queries without sequential search. The performance of this machine is mainly a function of desired accuracy, but has little to do with the size or depth of the search space. This result seems amazing, at least theoretically. In the practice, however, the Boltzmann machine trades accuracy for time,[13] and the harder part is not to get an answer from a

[13] According to Derthick, his sample problem takes about one minute to get the right answer most of the time. But to ensure 99.9% accuracy, the sample problem will take $10^{10^{10}}$ years to arrive at an answer.

constructed Boltzmann Machine network, but to select good patterns and appropriate weights. So far, the contribution of this approach is more a novel idea than a practical implementation.

6. Summary and Future Work

This report has identified a variety of opportunities to parallelize the operations of the Penman natural language generation system, especially for Nigel -- the grammar component of Penman not found in other software systems. Four different ways of processing the Nigel grammar were presented. These ranged from entering systems simultaneously to executing realization statements simultaneously. We analyzed each of these schemes, and we also indicated that a parallel Nigel computation is a natural implementation of Halliday's systemic model.

We have also analyzed the parallelism in KL-TWO, examined and assessed the two parallel approaches to KL-TWO by Forrest and Derthick.

A step beyond our preliminary investigation of parallelizing Penman will be to explore what kind of parallel machine is the most suitable for the implementation of the parallel schemes we have sketched. The answer to this question is by no means trivial since Penman is a complex system and it can be parallelized at many levels.

The level of parallelism we focus on will thus tend to determine what kind of machine is the best choice. For example, when multiple copies of Nigel and KL-TWO as well as multiple system processes are considered, each computation unit interacts with other units via messages only, and a message-passing hypercube multiprocessor would be a good candidate for achieving such computation behaviour. When the need for multiple choosers to have access to a common area (i.e., the Function Association Table) is taken into account, a shared-memory multiprocessor would seem to be better. In contrast, when we turn to Connectionist Models for KL-TWO, a data-level parallel machine such as the Connection Machine might be the best choice. However, such a machine will not be a good one for implementing our parallel model of computation within the Nigel component: our model does not exploit parallelism at the low level of the Connection Machine. In order to make use of the Connection Machine, we would need a low level model of parallelism such as a 'connectionist model' of systemic grammar comparable to the connectionist models developed for KL-TWO. While there would seem to be a suggestive correspondence between the idea of connectionism and the class of grammatical theories that interpret grammar as networks of relations, i.e. in particular Systemic Theory and Sidney Lamb's Stratificational theory, this has to be studied in detail.

Acknowledgements

We would like to express our gratitude to many of our colleagues at ISI, especially David Mizell and William C. Mann, for their inspiration and comments, and to Victor Brown for his help with the draft of this paper. We also thank Stephanie Forrest for her valuable comments on the draft. Needless to say, we are solely responsible for remaining errors and shortcomings.

References

[BrachmanLevesque 84]
Brachman, R. and Levesque, H.
The Tractability of Subsumption in Frame-Based Description Languages.
In *Proceedings of the National Conference on Artificial Intelligence*, pages 34-37. August, 1984.

[BrachmanSchmolze 85]
Brachman, R. and Schmolze, J.
An Overview of the KL-ONE Knowledge Representation System.
Cognitive Science :171-216, August, 1985.

[Derthick 86] Derthick, M.
A Connectionist Knowledge Representation System.
June, 1986.
Thesis Proposal, Department of Computer Science, Carnegie-Mellon University.

[FahlmanHinton 87]
Fahlman, S. and Hinton, G.
Connectionist Architectures for Artificial Intelligence.
Computer 20(1):100-109, January, 1987.

[Forrest 85] Forrest, S.
A study of Parallelism in the Classifier System and Its Application to Classification in KL-ONE Semantic Networks.
PhD thesis, University of Michigan, 1985.

[Forrest 87] Forrest, S.
Modeling High-level Symbolic Structures in Parallel Systems that Support Learning.
In *Proceedings of the 4th International Symposium on Modeling and Simulation Methodology*. January, 1987.

[Halliday 76] Halliday, M.
System and Function in Language.
Oxford University Press, London, 1976.

[Hinton 81] Hinton, G.
Implementing Semantic Networks in Parallel Hardware.
Parallel Models of Associative Memory.
Lawrence Erlbaum Associates, Publishers, 1981, Chapter 2.
edited by Hinton, G. and Anderson J.

[Holland 86] Holland, J., Holyoak, K., Nisbett, R. and Thagard, P.
Induction.
The MIT Press, 1986.

[KaczmarekBatesRobins 86]
Kaczmarek, T., Bates, R. and Robins, G.
Recent Developments in NIKL.
In *AAAI-86, Proceedings of the National Conference on Artificial Intelligence*. AAAI, Philadelphia, PA, August, 1986.

[KaczmarekMarkSondheimer 83]
Kaczmarek, T., Mark, W. and Sondheimer, N.
The Consul/CUE Interface: An Integrated Interactive Environment.
In *Proceedings of CHI '83 Human Factors in Computing Systems*, pages 98-102. ACM, December, 1983.

[Mann 83a] Mann, W.
An Overview of the Penman Text Generation System.
Technical Report ISI/RR-83-114, USC/Information Sciences Institute, April, 1983.

[Mann 83b] Mann, W.
Inquiry Semantics: A Functional Semanrics of Natural Language Grammar.
In *Proceedings of the First Annual Conference of the Association for Computational Linguistics*. Pisa, Italy, September, 1983.

[MannMatthiessen 83]
Mann, W. and Matthiessen, C.
Nigel: A Systemic Grammar for Text Generation.
Technical Report ISI/RR-83-105, USC/Information Sciences Institute, Feb, 1983.

[Matthiessen 87] Matthiessen, C.
Notes on the Organization of the Environment of a Text Generation Grammar.
In Kempen, G. (editor), *Natural Language Generation: Recent Advances in Artificial Intelligence, Psychology, and Linguistics.* Kluwer Academic Publishers, Boston/Dordrecht, 1987.

[Matthiessen 88] Matthiessen, C.
Semantics for a Systemic Grammar: The Chooser and Inquiry Framework.
In Benson, J., M. Cummings, W. Greaves (editor), *Systemic Perspectives on Discourse*. Benjamins, 1988.

[McAllester 82] McAllester, D.
Reasoning Utility Package User's Manual.
Technical Report, Massachusetts Institute Technology , April, 1982.

[SchmolzeLipkis 83]
Schmolze, J. and Lipkis, T.
Classification in the KL-ONE Knowledge Representation System.
In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* IJCAI, 1983.

[SondheimerNebel 86]
Sondheimer, N. and Nebel, B.
A Logical-Form and Knowledge-Base Design for Natural Language Generation.
In *AAAI-86, Proceedings of the National Conference on Artificial Intelligence.* AAAI, Philadelphia, PA, August, 1986.

[Vilain 85] Vilain, M.
The Restricted Language Architecture of a Hybrid Representation System.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 547-551. Los Angeles, CA, August, 1985.